Reasoning with Forest Logic Programs Using Fully Enriched Automata[⋆]

Cristina Feier[1] and Thomas Eiter[2]

[1] Department of Computer Science, University of Oxford, Oxford, UK
[2] Institute of Information Systems, Vienna University of Technology, Vienna, Austria

Abstract. Forest Logic Programs (FoLP) are a decidable fragment of Open Answer Set Programming (OASP) which have the forest model property. OASP extends Answer Set Programming (ASP) with open domains—a feature which makes it possible for FoLPs to simulate reasoning with the description logic $\mathcal{SHOQ}$. In the past, several tableau algorithms have been devised to reason with FoLPs, the most recent of which established a NEXPTIME upper bound for reasoning with the fragment. While known to be EXPTIME-hard, the exact complexity characterization of reasoning with FoLPs was still unknown. In this paper we settle this open question by a reduction of reasoning with FoLPs to emptiness checking of fully enriched automata which are known to be EXPTIME-complete.

1 Introduction

Open Answer Set Programming (OASP) [8] extends (function-free) Answer Set Programming (ASP) [5] with an open domain semantics: programs are interpreted with respect to arbitrary domains that might contain individuals which do not occur explicitly in the program. This enables to state generic knowledge using OASP; at the same time, OASP inherits from ASP the negation under the stable model semantics.

While OASP is undecidable in general, several decidable fragments have been found by restricting the shape of the rules. One such fragment are Forest Logic Programs (FoLP), which enjoy the forest model property: a unary predicate is satisfiable iff it is satisfied by a model representable as a labeled forest. FoLPs are quite expressive; e.g., one can simulate satisfiability testing of an ontology in the Description Logic (DL) $\mathcal{SHOQ}$ by them [4]. This led to f-hybrid KBs, which combine rules and ontologies distinctly from other approaches like dl-safe rules [9], r-hybrid knowledge bases [10], or MKNF$^+$ knowledge bases, as the interaction between the signatures of the two components is not restricted.

The simulation of $\mathcal{SHOQ}$ implies that reasoning with FoLPs is EXPTIME-hard; however, the exact complexity was open. A tableau-based algorithm in [4] gave an 2NEXPTIME upper bound, which an improved algorithm in [2] lowered to NEXPTIME. In this paper, we close this gap and show that deciding satisfiability of unary predicates w.r.t. FoLPs is EXPTIME-complete, by reducing emptiness checking of Fully Enriched Automata (FEAs) to this problem; hence, adding FoLP rules to SHOQ ontologies does not make reasoning harder. An extended version of the paper can be found at http://www.kr.tuwien.ac.at/research/reports/rr1502.pdf.

[⋆] Work supported by the EPSRC grants Score! and DBOnto and the FWF grant P24090.

2 Preliminaries

We assume countably infinite disjoint sets of constants, variables, and predicate symbols of positive arity. Terms and atoms are as usual. Atoms $p(\vec{t})$ are unary (resp. binary) if p is unary (resp. binary). A *literal* is an atom a or a negated atom $not\ a$. *Inequality literals* are of form $s \neq t$, where s and t are terms; all other literals are *regular*. For a set S of literals or (possibly negated) predicates, $S^+ = \{a \mid a \in S\}$ and $S^- = \{a \mid not\ a \in S\}$. If S is a set of (possibly negated) predicates of arity n and $\vec{t}$ are terms, then $S(\vec{t}) = \{l(\vec{t}) \mid l \in S\}$. For a set S of atoms, $not\ S = \{not\ a \mid a \in S\}$. A *program* is a countable set P of rules $r : \alpha \leftarrow \beta$, where α is a finite set of regular literals and β is a finite set of literals. We denote as $head(r)$ the set α, where α stands for a disjunction, and as $body(r)$ the set β, where β stands for a conjunction.

For R a rule, program, etc., let $vars(R)$, $preds(R)$, and $cts(R)$ be the sets of variables, predicates, and constants that occur in R, resp. A *universe* U for a program P is a non-empty countable set $U \supseteq cts(P)$. We let P_U be the grounding of P with U and let $\mathcal{B}_P$ be the set of regular atoms that can be formed from a ground program P.

An *interpretation* of a ground, i.e. variable free, program P is a subset I of $\mathcal{B}_P$. We write $I \models p(\vec{t})$ if $p(\vec{t}) \in I$ and $I \models not\ p(\vec{t})$ if $I \not\models p(\vec{t})$. For ground terms s, t, we write $I \models s \neq t$ if $s \neq t$. For a set of ground literals L, $I \models L$ if $I \models l$ for every $l \in L$. A ground rule $r : \alpha \leftarrow \beta$ is *satisfied* w.r.t. I, denoted $I \models r$, if $I \models l$ for some $l \in \alpha$ whenever $I \models \beta$. An interpretation I of a positive (i.e. not -free) ground program P is a *model* of P if I satisfies every rule in P; it is an *answer set* of P if it is a $\subseteq$- minimal model of P. For ground programs P with not, I is an answer set of P iff I is an answer set of $P^I = \{\alpha^+ \leftarrow \beta^+ \mid \alpha \leftarrow \beta \in P, I \models not\ \beta^-, I \models \alpha^-\}$.

An *open interpretation* of a program P is a pair (U, M) where U is a universe for P and M is an interpretation of P_U. An *open answer set* of P is an open interpretation (U, M) of P, with M an answer set of P_U.

Trees and Forests. Let $\mathbb{N}^+$ be the set of positive integers, and let $\langle \mathbb{N}^+ \rangle$ be the set of all sequences over $\mathbb{N}^+$, where ε is the empty sequence: for a sequence of constants and/or natural numbers s, $s \cdot \varepsilon = c$, where $\cdot$ is concatenation; also, by convention, $s \cdot c \cdot -1 = s \cdot c$, where c is a natural number, and $\varepsilon \cdot -1$ is undefined. A *tree* T with root c, also denoted as T_c, is a set of nodes, where each node is a sequence $c \cdot s$, where $s \in \langle \mathbb{N}^+ \rangle$, and for every $x \cdot d \in T_c$, $d \in \mathbb{N}^+$, $x \in T_c$. If c is irrelevant, we refer to T_c as T. Given a tree T, its arc set is $A_T = \{(x, y) \mid x, y \in T, \exists n \in \mathbb{N}^+ . y = x \cdot n\}$. We denote with $succ_T(x) = \{y \in T \mid y = x \cdot i, i \in N^+\}$ the successors of a node x in T and with $prec_T(x) = y$, where $x = y \cdot i \in T$, its predecessor.

A *forest* F is a set of trees $\{T_c \mid c \in C\}$, where C is a finite set of arbitrary constants. Its node set is $N_F = \cup_{T \in F} T$ and its arc set is $A_F = \cup_{T \in F} A_T$. For a node $x \in N_F$, $succ_F(x) = succ_T(x)$, and $prec_F(x) = prec_T(x)$, where $x \in T$ and $T \in F$. For a node $y = x \cdot i \in T$ and $T \in F$, $prec_F(y) = prec_T(y) = x$. An *interconnected forest* EF is a tuple (F, ES), where $F = \{T_c \mid c \in C\}$ is a forest and $ES \subseteq N_F \times C$. Its set of nodes is $N_{EF} = N_F$, and its set of arcs is $A_{EF} = A_F \cup ES$. A Σ-labelled forest is a tuple (F, f) where F is an interconnected forest/tree and $f : N_F \to \Sigma$ is a labelling function, where Σ is any set of symbols.

3 Forest Logic Programs

Forest Logic Programs (FoLPs) are a fragment of OASP which have the forest model property. They allow only for unary and binary predicates and tree-shaped rules.

Definition 1. *A* forest logic program (FoLP) *is an OASP with only unary and binary predicates, s.t. a rule is either:*

- *a* free rule*:* $a(s) \vee not\ a(s) \leftarrow$ (1) *or* $f(s,t) \vee not\, f(s,t) \leftarrow$ (2)
- *a unary rule:* $a(s) \leftarrow \beta(s), \gamma_1(s,t_1), \ldots, \gamma_m(s,t_m), \delta_1(t_1), \ldots, \delta_m(t_m), \psi$ (3), *with* $\psi \subseteq \{t_i \neq t_j | 1 \leqslant i < j \leqslant m\}$ *and* $m \in \mathbb{N}$,
- *or a* binary rule*:* $f(s,t) \leftarrow \beta(s), \gamma(s,t), \delta(t)$ (4),

where a is a unary predicate, and f is a binary predicate; s, t, and t_i-s are distinct terms; β, δ, and δ_i-s are sets of (possibly negated) unary predicates; γ, and γ_i-s are sets of (possibly negated) binary predicates; inequality does not appear in γ and γ_i; $\gamma_i^+ \neq \emptyset$, if t_i is a variable, for every $1 \leqslant i \leqslant m$, and $\gamma^+ \neq \emptyset$, if t is a variable.

A predicate q in a FoLP P is *free* if it occurs in a free rule in P. We denote with $upr(P)$, and $bpr(P)$ (resp. $urul(P)$, and $brul(P)$), the sets of unary and binary predicates (resp. unary and binary rules) which occur in P. The degree of a unary rule r of type (3), denoted $degree(r)$, is the number k of successor variables appearing in r. The degree of a free rule is 0. The degree of a FoLP P is $degree(P) = \sum_{p \in upr(P)} degree(p)$, where $degree(p) = max\{degree(r) \mid p \in preds(head(r))\}$.

A forest model of an OASP P that satisfies a unary predicate p is a forest which contains for each constant in P a tree having the constant as root, and possibly one more tree with an anonymous root; the predicate p is in the label of some root node.

Definition 2. *Let P be a program. A predicate $p \in upr(P)$ is* forest satisfiable *w.r.t. P if there exist an open answer set (U, M) of P; an interconnected forest $EF = (\{T_\rho\} \cup \{T_a \mid a \in cts(P)\}, ES)$, where ρ is a constant, possibly from $cts(P)$; and a labelling function $ef : \{T_\rho\} \cup \{T_a \mid a \in cts(P)\} \cup A_{EF} \rightarrow 2^{preds(P)}$ s. t. $p \in ef(\rho)$; $U = N_{EF}$; $ef(x) \in 2^{upr(P)}$, when $x \in T_\rho \cup \{T_a \mid a \in cts(P)\}$; $ef(x) \in 2^{bpr(P)}$, when $x \in A_{T_\rho}$; $M = \{p(x) \mid x \in N_{EF}, p \in ef(x)\} \cup \{f(x,y) \mid (x,y) \in A_{EF}, f \in ef(x,y)\}$; and for every $(z, z \cdot i) \in A_{EF}$: $ef(z, z \cdot i) \neq \emptyset$. We call such a pair (U, M) a* forest model.

P has the forest model property *if every unary predicate p that is satisfiable w.r.t. P, is forest satisfiable w.r.t. P; FoLPs enjoy this property [7].*

4 Fully Enriched Automata

Fully enriched automata (FEAs) were introduced in [1] as a tool to reason in hybrid graded μ-calculus. They accept forests as input. We describe them following [1].

For a set Y, we denote with $B^+(Y)$ the set of positive Boolean formulas over Y, where $true$ and $false$ are also allowed and where $\wedge$ has precedence over $\vee$. For a set $X \subseteq Y$ and a formula $\theta \in B^+(Y)$, we say that X satisfies θ iff assigning true to elements in X and assigning false to elements in $Y - X$ makes θ true. For $b > 0$, let $D_b = \{\langle 0 \rangle, \langle 1 \rangle, \ldots, \langle b \rangle\} \cup \{[0], [1], \ldots, [b]\} \cup \{-1, \varepsilon, \langle root \rangle, [root]\}$.

A fully enriched automaton (FEA) is a tuple $A = \langle \Sigma, b, Q, \delta, q_0, \mathcal{F} \rangle$, where Σ is a finite input alphabet, $b > 0$ is a counting bound, Q is a finite set of states, δ :

$Q \times \Sigma \to B^+(D_b \times Q)$ is a transition function, $q_0 \in Q$ is an initial state, and $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_k\}$, where $\mathcal{F}_1 \subseteq \mathcal{F}_2 \subseteq \ldots \subseteq \mathcal{F}_k = Q$ is a *parity acceptance condition*. The number k of sets in $\mathcal{F}$ is the *index* of the automaton.

A *run* of a FEA on a labeled forest (F, V) is an $N_F \times Q$-labeled tree (T_c, r) s.t. $r(c) = (d, q_0)$, for some root d in F, and for all $y \in T_c$ with $r(y) = (x, q)$ and $\delta(q, V(x)) = \theta$, there is a (possibly empty) set $S \subseteq D_b \times Q$ such that S satisfies θ and for all $(d, s) \in S$, the following hold: *(i)* if $d \in \{-1, \varepsilon\}$, then $x \cdot d$ is defined and there is $j \in \mathbb{N}^+$ such that $y \cdot j \in T_c$ and $r(y \cdot j) = (x \cdot d, s)$; *(ii)* if $d = \langle n \rangle$, then there is a set $M \subseteq succ_F(x)$ of cardinality $n + 1$ s.t. for all $z \in M$, there is $j \in \mathbb{N}^+$ s.t. $y \cdot j \in T_c$ and $r(y \cdot j) = (z, s)$; *(iii)* if $d = [n]$, then there is a set $M \subseteq succ_F(x)$ of cardinality n s.t. for all $z \in succ_F(x) - M$, there is $j \in \mathbb{N}^+$ s.t. $y \cdot j \in T_c$ and $r(y \cdot j) = (z, s)$; *(iv)* if $d = \langle root \rangle$ ($d = [root]$), then for some (all) root(s) $c \in F$ there exists $j \in \mathbb{N}^+$ s.t. $y \cdot j \in T_c$ and $r(y \cdot j) = (c, s)$;

If θ above is true, then y does not need to have successors. Moreover, since no set S satisfies $\theta = false$, there cannot be any run that takes a transition with $\theta = false$. A run is *accepting* if each of its infinite paths π is accepting, that is if the minimum i for which $Inf(\pi) \cap \mathcal{F}_i \neq \emptyset$, where $Inf(\pi)$ is the set of states occurring infinitely often in π, is even. The automaton accepts a forest iff there exists an accepting run of the automaton on the forest. The language of A, denoted $\mathcal{L}(A)$, is the set of forests accepted by A. We say that A is non-empty if $\mathcal{L}(A) \neq \emptyset$.

Theorem 1 (Corollary 4.3 [1]). *Given a FEA $A = \langle \Sigma, b, Q, \delta, q_0, \mathcal{F} \rangle$ with n states and index k, deciding whether $\mathcal{L}(A) = \emptyset$ is possible in time $(b+2)^{\mathcal{O}(n^3 \cdot k^2 \cdot \log k \cdot \log b^2)}$.*

5 From Forest Logic Programs to Fully Enriched Automata

In this section we reduce satisfiability checking of unary predicates w.r.t. FoLPs to emptiness checking for FEAs. For a FoLP P and a unary predicate p, we introduce a class of FEAs $A^{p,P}_{\rho,\theta}$, where ρ is one of $cts(P)$ or a new anonymous individual and $\theta : cts(P) \cup \{\rho\} \to 2^{upr(P) \cup cts(P) \cup \{\rho\}}$ is s.t. $o_i \in \theta(o_i)$, and $o_j \notin \theta(o_i)$, for every $o_i, o_j \in cts(P) \cup \{\rho\}$, s.t. $o_i \neq o_j$. Furthermore, $p \in \theta(c)$, where c is one of $cts(P) \cup \{\rho\}$ and c is ρ if $\rho \notin cts(P)$. Intuitively, $A^{p,P}_{\rho,\theta}$ accepts forest models of p w.r.t. P encoded in a certain fashion: for every root in the forest model, the root node will appear in its own label; function θ fixes a content for the label of each root of accepted forest models.

Let $d = degree(P)$ and let $\textsc{Pat}_P = \{*\} \cup cts(P)$ be the set of *term patterns*, where $*$ stands for a generic anonymous individual: a term t matches a term pattern pt, written $t \mapsto pt$, iff $t = pt$, when t is a constant; if t is not a constant, the match trivially holds. We use term patterns as a unification mechanism: a variable matches with a constant or an anonymous individual, but a constant matches only with itself. $A^{p,P}_{\rho,\theta}$ will run on forests labelled using the following alphabet: $\Sigma = 2^S$, where $S = upr(P) \cup \{1, \ldots, d\} \cup cts(P) \cup \{\rho\} \cup \{\uparrow^o_f \mid f \in bpr(P)\} \cup \{\downarrow^t_f \mid f \in bpr(P), t \in \textsc{Pat}_P\}$.

Unlike forest models, arcs of forests accepted by FEAs are not labelled: as such, binary predicates occur in the label of nodes in an adorned form. These adorned predicates are of form $\downarrow^t_f$, in which case they represent an f-link between the predecessor of the labelled node, which has term pattern t and the node itself, or of form $\uparrow^o_f$, in which case the current node is linked to a constant o from P via the binary predicate f. Besides unary predicates, labels might contain natural numbers and constants, which will be used as an addressing mechanism for successors of a given node and nodes which stand for constants in accepted forests, resp. The set of states of the automaton are as follows: $Q = Q_i \cup Q_+ \cup Q_-$, with:

- $Q_i = \{q_0, q_1\} \cup \{q_o \mid o \in cts(P) \cup \{\rho\}\} \cup \{q_{\neg k} \mid 1 \leqslant k \leqslant d\}$,
- $Q_+ = \{q_{t,a}, q_{t,r_a}, q_{t_1,t_2,u}, q_{t_1,t_2,r_f}, q_{k,t,*,u} \mid t, t_1, t_2 \in \textsc{Pat}_P, a \in upr(P), f \in bpr(P), u$ is of form $a, f, not\ a$ or $not\ f, 1 \leqslant k \leqslant d, r_a \in urul(P), r_f \in brul(P)\}$,
- $Q_- = \{q_{\overline{t,a}}, q_{\overline{t,r_a}}, q_{\overline{t_1,t_2,u}}, q_{\overline{t_1,t_2,r_f}}, q_{\overline{k,t,*,u}} \mid t_1, t_2, t, a, f, u, k, r_a, r_f$ as above$\}$.

Positive states in Q_+ (resp. negative states in Q_-) are used to motivate the presence (resp. absence) of atoms in an open answer set. Q_i contains q_0, the initial state, q_1, a state which will be visited recursively in every node of the forest, q_o, a state corresponding to the initial visit of each constant node, and $q_{\neg k}$, a state which asserts that for every node in an accepted forest there must be at most one successor which has k in the label.

We next describe the transition function of $A^{p,P}_{\rho,\theta}$. The initial transition prescribes that the automaton visits a root of the forest in state q_o, for every $o \in cts(P) \cup \{\rho\}$:

$$\delta(q_0, \sigma) = \bigwedge_{o \in cts(P) \cup \{\rho\}} (\langle root \rangle, q_o) \tag{5}$$

In every such state q_o, it should hold that o and only o is part of the label. Furthermore, the automaton justifies the presence and absence of each unary predicate a and adorned upward binary predicate in the label[3] by entering states $q_{o,a}$, $q_{o,o',f}$, $q_{\overline{o,a}}$, and $q_{\overline{o,o',f}}$ resp. At the same time every successor of the constant node is visited in state q_1:

$$\begin{aligned}\delta(q_o, \sigma) = o \in \sigma \wedge \textstyle\bigwedge_{o' \in cts(P) \cup \{\rho\} - \{o\}} o' \notin \sigma \wedge \bigwedge_{a \in \theta(o)} (\varepsilon, q_{o,a}) \wedge \bigwedge_{a \notin \theta(o)} (\varepsilon, q_{\overline{o,a}}) \\ \wedge \textstyle\bigwedge_{\uparrow^{o'}_f \in \theta(o)} (\varepsilon, q_{o,o',f}) \wedge \bigwedge_{\uparrow^{o'}_f \notin \theta(o)} (\varepsilon, q_{\overline{o,o',f}}) \wedge ([0], q_1)\end{aligned} \tag{6}$$

Whenever the automaton finds itself in state q_1 it tries to motivate the presence and absence of each unary and each adorned binary predicate in its label and then it recursively enters the same state into each successor of the current node:

$$\begin{aligned}\delta(q_1, \sigma) = \textstyle\bigwedge_{a \in \sigma} (\varepsilon, q_{*,a}) \wedge \bigwedge_{a \notin \sigma} (\varepsilon, q_{\overline{*,a}}) \wedge \bigwedge_{\downarrow^t_f \in \sigma} (\varepsilon, q_{t,*,f}) \wedge \bigwedge_{\downarrow^t_f \notin \sigma} (\varepsilon, q_{\overline{t,*,f}}) \wedge \\ \textstyle\bigwedge_{\uparrow^{o'}_f \in \sigma} (\varepsilon, q_{*,o',f}) \wedge \bigwedge_{\uparrow^{o'}_f \notin \sigma} (\varepsilon, q_{\overline{*,o',f}}) \wedge ([0], q_1) \wedge \bigwedge_{1 \leqslant k \leqslant d} ([1], q_{\neg k})\end{aligned} \tag{7}$$

It also ensures that for each integer $1 \leqslant k \leqslant d$, the labels of each but one successor do not contain k:

$$\delta(q_{\neg k}, \sigma) = k \notin \sigma \tag{8}$$

To motivate a predicate in a node label, the automaton checks whether it is free (using a test $free(.)$) or finds a supporting rule. We distinguish between unary and binary predicates and the term pattern for the node where the predicate has to hold. For unary predicates holding at a constant node, a first check is that we are at the right node - this is needed as later the automaton will visit all roots in this state. For binary predicates, depending on the term pattern, the label is checked for different types of adorned binary atoms.

$$\delta(q_{*,a}, \sigma) = a \in \sigma \wedge \Big(free(a) \vee \bigvee_{r_a : a(s) \leftarrow \beta \in P} (\varepsilon, q_{*,r_a}) \Big) \tag{9}$$

[3] Constants have no predecessors, hence there are no adorned downward predicates in the label.

$$\delta(q_{o,a}, \sigma) = o \notin \sigma \vee a \in \theta(o) \wedge \Big(free(a) \vee \bigvee_{r_a : a(s) \leftarrow \beta \in P, s \mapsto o} (\varepsilon, q_{o,r_a})\Big) \tag{10}$$

$$\delta(q_{t,*,f}, \sigma) = \ \downarrow_f^t \in \sigma \wedge \Big(free(f) \vee \bigvee_{r_f : f(s,v) \leftarrow \beta \in P, s \mapsto t, v \mapsto *} (\varepsilon, q_{v,*,r_f})\Big) \tag{11}$$

$$\delta(q_{t,o,f}, \sigma) = \ \uparrow_f^o \in \sigma \wedge \Big(free(f) \vee \bigvee_{r_f : f(s,v) \leftarrow \beta \in P, s \mapsto t, v \mapsto o} (\varepsilon, q_{t,o,r_f})\Big) \tag{12}$$

Let $r_a : a(s) \leftarrow \beta(s), (\gamma_i(s, v_i), \delta_i(v_i))_{1 \leqslant i \leqslant m}, \psi$ be a unary rule. Then, we denote with J_{r_a} a multiset $\{j_i \mid 1 \leqslant i \leqslant m, j_i \in \{1, \ldots, d\} \cup cts(P)\}$ such that for every $j_i \in J_{r_a}$, $v_i \in cts(P)$ implies $j_i = v_i$, and for every $j_i, j_l \in J_{r_a}$, $v_i \neq v_l \in \psi$ implies $j_i \neq j_l$. A multiset provides a way to satisfy the successor part of a unary rule in a forest model by identifying the successor terms of the rule with successors of the current element in the model or constants in the program. Let $\mathcal{MJ}$ be the set of all such multisets. The following transition describes how the body of such a rule is checked to be satisfiable:

$$\begin{aligned}\delta(q_{t,r_a}, \sigma) = \bigwedge_{u \in \beta} (\varepsilon, q_{*,t,u}) \wedge \bigvee_{J_{r_a} \in \mathcal{MJ}} \Big(\bigwedge_{k=1}^{d} \bigwedge_{j_i = k, j_i \in J_{r_a}} \bigwedge_{u \in \gamma_i \cup \delta_i} (\langle 0 \rangle, q_{k,t,*,u}) \wedge \\ \bigwedge_{o \in cts(P)} \bigwedge_{j_i = o, j_i \in J_{r_a}} \bigwedge_{u \in \gamma_i \cup \delta_i} (\varepsilon, q_{t,o,u}) \Big)\end{aligned} \tag{13}$$

State $q_{k,t,*,u}$ checks that the (possibly negated) unary or adorned binary predicate u is (is not) part of the label of the k-th successor of a given node:

$$\delta(q_{k,t_1,t_2,u}, \sigma) = k \in \sigma \wedge \bigwedge_{j \neq k} j \notin \sigma \wedge (\varepsilon, q_{t_1,t_2,u}) \tag{14}$$

State $q_{t_1,t_2,u}$ can be seen as a multi-state with different transitions depending on its arguments (two transitions have already been introduced as rules (11) and (12) above): if $t_2 = *$, one has the justify the presence/absence of u in the label of the current node; otherwise, when $t_2 = o$, one has to justify it from the label of the root node corresponding to constant o: note that, as it is not possible to jump directly to a given root node in the forest, nor to enforce that there will be a single root node corresponding to each constant, in transition (17) we visit each root node in state $q_{o,a}$:

$\delta(q_{t_1,t_2,u}, \sigma) =$

$(\varepsilon, q_{*,a})$,	if $t_2 = * \wedge u = a$ (15)	$a \notin \theta(o)$,	if $t_2 = o \wedge u = not\ a$ (16)
$([root], q_{o,a})$,	if $t_2 = o \wedge u = a$ (17)	$\downarrow_f^t \notin \sigma$,	if $t_2 = * \wedge u = not\ f$ (18)
$a \notin \sigma$,	if $t_2 = * \wedge u = not\ a$ (19)	$\uparrow_f^o \notin \theta(o)$,	if $t_2 = o \wedge u = not\ f$ (20)

For binary rules: $r_f : f(s,v) \leftarrow \beta(s), \gamma(s,v), \delta(v)$, where v is grounded using an anonymous individual, we also look at the predecessor node to see if the local part of the rule is satisfied. When v is grounded using a constant, the local part of the rule is checked at the current node and the successor part at the respective constant.

$$\delta(q_{t,*,r_f}, \sigma) = \bigwedge_{u \in \beta} (-1, q_{*,t,u}) \wedge \bigwedge_{u \in \gamma \cup \delta} (\varepsilon, q_{t,*,u}) \tag{21}$$

$$\delta(q_{t,o,r_f}, \sigma) = \bigwedge_{u \in \beta} (\varepsilon, q_{*,t,u}) \wedge \bigwedge_{u \in \gamma \cup \delta} (\varepsilon, q_{t,o,u}) \tag{22}$$

The transitions of the automaton in the negative states can be seen as dual versions of the ones for the counterpart positive states. They are presented in the technical report.

Finally we specify the parity acceptance condition. The index of the automaton is 2, with $\mathcal{F}_1 = \{q_{t,a}, q_{t_1,t_2,f} \mid a \in upr(P), f \in bpr(P), t, t_1, t_2 \in \textsc{Pat}_P\}$ and $\mathcal{F}_2 = Q$. Intuitively, paths in a run of the automaton correspond to dependencies of literals in the accepted model and by disallowing the infinite occurrence on a path of states associated to atoms in the model we ensure that only well-supported models are accepted.

Theorem 2. *Let P be a FoLP and p be a unary predicate symbol. Then, p is satisfiable w.r.t. P iff there exists an automaton $A^{p,P}_{\rho,\theta}$ such that $\mathcal{L}(A^{p,P}_{\rho,\theta}) \neq \emptyset$.*

Theorem 3. *Satisfiability checking of unary predicates w.r.t. FoLPs is* EXPTIME*-complete.*

6 Discussion and Conclusion

We have described a reduction of the satisfiability checking task of unary predicates w.r.t. FoLPs to emptiness checking of FEAs. This enabled us to establish a tight complexity bound on this reasoning task for FoLPs. Other reasoning tasks like consistency checking of FoLPs and skeptical and brave entailment of ground atoms can be polynomially reduced to satisfiability checking of unary predicates [6]; thus, the complexity result applies to those tasks as well. Also, by virtue of the translation from fKBs to FoLPs, the result applies to fKBs as well: satisfiability checking of unary predicates w.r.t. fKBs is EXPTIME-complete. Thus, reasoning with FoLP rules and $\mathcal{SHOQ}$ ontologies is not harder than reasoning with $\mathcal{SHOQ}$ ontologies themselves.

Finally, as our result shows that FoLPs have the same complexity as CoLPs, we plan to further investigate the extension of the deterministic AND/OR tableau algorithm for CoLPs [3] to FoLPs. As explained in [3], such an extension is far from trivial.

References

1. Piero A. Bonatti, Carsten Lutz, Aniello Murano, and Moshe Y. Vardi. The complexity of enriched μ-calculi. *Logical Methods in Computer Science*, 4(3):1–27, 2008.
2. C. Feier. Worst-case optimal reasoning with Forest Logic Programs. In *Proc. KR 2012*, pages 208–212, 2012.
3. C. Feier. *Reasoning with Forest Logic Programs*. PhD thesis, TU Wien, 2014.
4. C. Feier and S. Heymans. Reasoning with Forest Logic Programs and f-hybrid knowledge bases. *TPLP*, 3(13):395–463, 2013.
5. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of ICLP'88*, pages 1070–1080, 1988.
6. S. Heymans. *Decidable Open Answer Set Programming*. PhD thesis, Theoretical Computer Science Lab (TINF), Department of Computer Science, Vrije Universiteit Brussel, 2006.
7. S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Open Answer Set Programming for the Semantic Web. *Journal of Applied Logic*, 5(1):144–169, 2007.
8. S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Open Answer Set Programming with guarded programs. *Transactions on Computational Logic*, 9(4):1–53, August 2008.
9. B. Motik, U. Sattler, and R. Studer. Query answering for OWL-DL with rules. *Journal of Web Semantics*, 3(1):41–60, July 2005.
10. R. Rosati. On combining description logic ontologies and nonrecursive datalog rules. In *Proc. RR*, pages 13–27, 2008.